Московский авиационный институт
(государственный технический университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа №3

по курсу
«Логическое программирование»

«Применение Пролога для решения задач поиска в пространстве состояний»

Выполнила: Карпова В.А.
Группа: 08-306
№ по списку: 9
Руководитель: Левинская М.А.
Оценка:
Дата:

Москва
2012 г.

Задание

Написать и отладить Пролог-программу решения задачи искусственного интеллекта, используя технологию поиска в пространстве состояний, в соответствии с номером варианта.

Вариант 2

Три миссионера и три каннибала хотят переправиться с левого берега реки на правый. Как это сделать за минимальное число шагов, если в их распоряжении имеется трехместная лодка и ни при каких обстоятельствах (в лодке или на берегу) миссионеры не должны оставаться в меньшинстве.

Метод решения

```
  % Начальное состояние
  start([[m,m,m], [k,k,k], [], []]).

  % Финальное состояние
  finish([[], [], [m,m,m], [k,k,k]]).

  % Проверка на непротиворечивость утверждения: миссионеры не
остаются в меньшинстве (на берегу)
  test_land([Lm, Lk, Rm, Rk]) :-
     length(Lm, L1),
     length(Lk, L2),
     length(Rm, R1),
     length(Rk, R2),
     test_num(L1, L2, R1, R2).

  test_num(0, _, 0, _).

  test_num(0, _, R1, R2) :-
     R1 >= R2.

  test_num(L1, L2, 0, _) :-
     L1 >= L2.

  test_num(L1, L2, R1, R2) :-
     L1 >= L2,
     R1 >= R2.

  % Возможные случаи передвижения лодки (с учетом превышения
миссионеров в лодке)
  ship_left([Lm1, Lk1, Rm1, Rk1], [Lm2, Lk2, Rm2, Rk2]) :-
ship_right([Rm1, Rk1, Lm1, Lk1],[Rm2, Rk2, Lm2, Lk2] ).

  ship_right([[M|Lm], Lk, Rm, Rk], S2) :-
        S2 = [Lm, Lk, [M|Rm], Rk],      test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  ship_right([[M1,M2|Lm], Lk, Rm, Rk], S2) :-
        S2 = [Lm, Lk, [M1,M2|Rm], Rk],       test_land(S2),
test_land([Lm, Lk, Rm, Rk]).
```

```
  ship_right([[M1,M2,M3|Lm], Lk, Rm, Rk], S2) :-
        S2 = [Lm, Lk, [M1,M2,M3|Rm], Rk],        test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  ship_right([Lm, [K1|Lk], Rm, Rk], S2) :-
        S2 = [Lm, Lk, Rm, [K1|Rk]],         test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  ship_right([Lm, [K1,K2|Lk], Rm, Rk], S2) :-
        S2 = [Lm, Lk, Rm, [K1,K2|Rk]],      test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  ship_right([Lm, [K1,K2,K3|Lk], Rm, Rk], S2) :-
        S2 = [Lm, Lk, Rm, [K1,K2,K3|Rk]],        test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  ship_right([[M|Lm], [K|Lk], Rm, Rk], S2) :-
        S2 = [Lm, Lk, [M|Rm], [K|Rk]],      test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  ship_right([[M1,M2|Lm], [K|Lk], Rm, Rk], S2) :-
        S2 = [Lm, Lk, [M1,M2|Rm], [K|Rk]],  test_land(S2),
test_land([Lm, Lk, Rm, Rk]).

  %Совершить одно передвижение лодки и проверить
  move(S1, S2) :- ship_right(S1, X), ship_left(X, S2), S1 \= S2.

  % Окончание решения
  final(S) :- ship_right(S, F), finish(F).

  % Продление пути [Temp|Tail] всеми возмождными способами, не
приводящими к зацикливанию
  prolong([Temp|Tail],[New,Temp|Tail]):-
          move(Temp,New),not(member(New,[Temp|Tail])).

  % DFS, Поиск в глубину
  dpth([Finish|Tail],[Finish|Tail]) :- final(Finish).
  dpth(TempWay,Way):-
          prolong(TempWay,NewWay),
          dpth(NewWay,Way).

  % BFS, Поиск в ширину
  bdth([[Finish|Tail]|_],[Finish|Tail]) :- final(Finish). %
Отвечает за нахождение подходящего решения.
  %Из очереди берется путь TempWay и ищутся с помощью предиката
findall всевозможные его продления (prolong),
  %Эти продлдения помещаются в список путей Ways. Ways добавляется
в конец хвоста исходной очереди OtherWays, в результате получается
очередь NewWays, к которой затем алгоритм поиска в ширину
применяется рекурсивно.
  bdth([TempWay|OtherWays],Way):-
          findall(W,prolong(TempWay,W),Ways),
          append(OtherWays,Ways,NewWays),
          bdth(NewWays,Way).
```

```
  % Iter
  % Поиск с итерационным заглублением. Ограничивает поиск в глубину
с ограничением по глубине.
  int(1).
  int(N):-int(M),N is M+1.

  search_iter(Start,Way):-
int(Lev),(Lev>100,!;id([Start],Way,Lev)).

  id([Finish|Tail],[Finish|Tail],0) :- final(Finish).
  id(TempWay,Way,N):-N>0,
          prolong(TempWay,NewWay),N1 is N-1,
          id(NewWay,Way,N1).

  % Поиск в глубину, ширину и с итерационным заглублением
  d3(Path) :- start(S), dpth([S], Path).
  b3(Path) :- start(S), bdth([[S]], Path).
  i3(Path) :- start(S), search_iter(S, Path).

  % Применение с распечаткой решения
  taste_dfs(X) :- d3(X),!, print_answer(X).
  taste_bfs(X) :- b3(X),!, print_answer(X).
  taste_iter(X) :- i3(X),!, print_answer(X).

  print_answer(X):-
     finish(S),
     on_land(X), nl, write(S),
     nl,nl,
     in_boat(X), nl.

  % Показать последовательность состояний на левом и правом берегу
  on_land([_]):-!.
  on_land([A,B|Tail]):-
          on_land([B|Tail]), nl, write(B), write('\n move right ->
'),
          ship_right(A,C), !, ship_left(C,B), nl, write(C),
write('\n move left <- ').

  check_count(M1, K1, M2, K2, ResM, ResK):-
     ResM is max(M1,M2),
     ResK is max(K1,K2).

  % Показать последовательность состояний в лодке
  show_boat([Lm1, Lk1, Rm1, Rk1], [Lm2, Lk2, Rm2, Rk2]):-
     length(Lm1,LenLm1), length(Lk1,LenLk1), length(Rm1,LenRm1),
length(Rk1,LenRk1),
     length(Lm2,LenLm2), length(Lk2,LenLk2), length(Rm2,LenRm2),
length(Rk2,LenRk2),
     check_count(LenLm1-LenLm2, LenLk1-LenLk2, LenRm1-LenRm2,
LenRk1-LenRk2, ResM, ResK),
     Res = [ResM, ResK],
     write(' boat: '), write(Res).

in_boat([_]):-!.
```

```
in_boat([A,B|Tail]):-
   in_boat([B|Tail]),
   ship_right(A,C), show_boat(B,C), nl, !,
   ship_left(C,B), show_boat(A,C), nl.
```

Результат

```
?- ['3_07.05.12.pro'].
% 3_07.05.12.pro compiled 0.00 sec, -144 bytes
true.

?- taste_dfs(_).

[[m, m, m], [k, k, k], [], []]
 move right ->
[[m, m], [k, k], [m], [k]]
 move left <-
[[m, m, m], [k, k], [], [k]]
 move right ->
[[], [k, k], [m, m, m], [k]]
 move left <-
[[m, m], [k, k], [m], [k]]
 move right ->
[[], [k, k], [m, m, m], [k]]
 move left <-
[[], [k], [m, m, m], [k, k]]
 move right <-
[[], [], [m, m, m], [k, k, k]]

 boat: [1, 1]
 boat: [1, 0]
 boat: [3, 0]
 boat: [2, 0]
 boat: [2, 0]
 boat: [0, 1]
 boat: [0, 1]

true .

?- taste_bfs(_).

[[m, m, m], [k, k, k], [], []]
 move right ->
[[m, m], [k, k], [m], [k]]
 move left <-
[[m, m, m], [k, k], [], [k]]
 move right ->
[[], [k, k], [m, m, m], [k]]
 move left <-
[[], [k], [m, m, m], [k, k]]
 move right <-
[[], [], [m, m, m], [k, k, k]]
```

```
 boat: [1, 1]
 boat: [1, 0]
 boat: [3, 0]
 boat: [0, 1]
 boat: [0, 1]

true .

?- taste_iter(_).

[[m, m, m], [k, k, k], [], []]
 move right ->
[[m, m], [k, k], [m], [k]]
 move left <-
[[m, m, m], [k, k], [], [k]]
 move right ->
[[], [k, k], [m, m, m], [k]]
 move left <-
[[], [k], [m, m, m], [k, k]]
 move right <-
[[], [], [m, m, m], [k, k, k]]

 boat: [1, 1]
 boat: [1, 0]
 boat: [3, 0]
 boat: [0, 1]
 boat: [0, 1]
```

Замечания

Поиск в глубину выдал не оптимальный метод решения. В то время как поиск в ширину и с итерационным заглублением выдали одинаковые оптимальные результаты. Рассмотрим основные плюсы и минусы каждого из методов поиска:

- Поиск в ширину:

Плюсы:

Находит оптимальное решение, т.к. если существует кратчайший путь в графе, то именно он будет найден быстрее всех

✓ Следующий удовлетворяющий путь часто находит быстрее, так как обычно на одной глубине их несколько

✓ Подходит для поиска кратчайшего пути в бесконечном графе

Минусы:

Чем больше найдено решений, тем дольше ищет следующие

✘ Обычно, выполняется дольше и потребляет больше памяти, чем поиск в глубину

- Для поиска в глубину:

Плюсы:

✓ Скорость поиска намного выше

✓ Поиск следующего решения по скорости не зависит от количества уже найденных

✓ Потребляет немного памяти

✓ Позволяет найти все решения, так как не превышает лимит памяти

Минусы:

- ✖ Не подходит для бесконечных графов
- ✖ Ищет решения, но не факт, что они будут оптимальными
- ✖ В случае нахождения решения в правом конце графа, поиск будет слишком долгий

- Итерационный поиск

Плюсы:

- ✓ Уменьшение затрат по памяти по сравнению с поиском в ширину
- ✓ Обладает всеми преимуществами поиска в глубину и в ширину

Минусы:

- ✖ Уменьшение эффективности (времени работы) за счет того, что найденные на предыдущем шаге меньшие пути забываются, и исследование пространства поиска начинается заново

Выводы

В данной лабораторной работе реализованы три метода поиска пути в графе (пространстве состояний) для решения поставленной задачи.

Уже при работе видно, что метод поиска в глубину будет менее прожорливым, а для этой задачи еще и быстрым, в отличие от поиска в ширину. Если бы отсутствовали циклические зависимости вершин графа, то можно было бы обойтись и без проверки на “повторность” вершины в пути.

Кроме того, поиск в глубину находит решение, которое сложно назвать оптимальным. Алгоритм не думает о коротком пути, здесь будет найдено достаточно длинное решение от начальной до завершающей вершины.

Так что, путь, полученный при поиске в ширину в этом плане лучше, ведь он будет самым коротким в силу особенности алгоритма. Но это потребует больших затрат из-за огромного числа путей, которые будут храниться в своеобразной очереди обработки.

Единственное что можно сделать для оптимизации расхода памяти, это воспользоваться древовидным представлением путей в очереди. Тогда их начальные участки, которые являются общими для нескольких, не будут дублироваться, и избыточность будет устранена.